Laser Sketch

Walker Chan, Sharon Chou, Huy Nguyen

6.111 Spring 2007

T.A.: David Wentzloff

May 17, 2007

Abstract

Laser Sketch is a vector-based drawing program that accepts user input from a mouse and keyboard. The program then displays the graphics on a VGA monitor and projects them on a screen with a laser scanner. The system is divided into three components: a user interface, a vector renderer / RAM, and a laser driver. The UI takes data from the PS/2 mouse clicks and keyboard presses, sending it to the vector renderer. The renderer then processes the data, transforms them into objects (either lines or circles), and sends them to the VGA frame buffer and laser driver. The renderer can also translate and rotate the objects. Finally, the laser driver formats the data from the renderer for outputting to DACs which drive the laser scanner. It also provides options for the user to adjust the images.

Table of Contents

1. High-level System Description

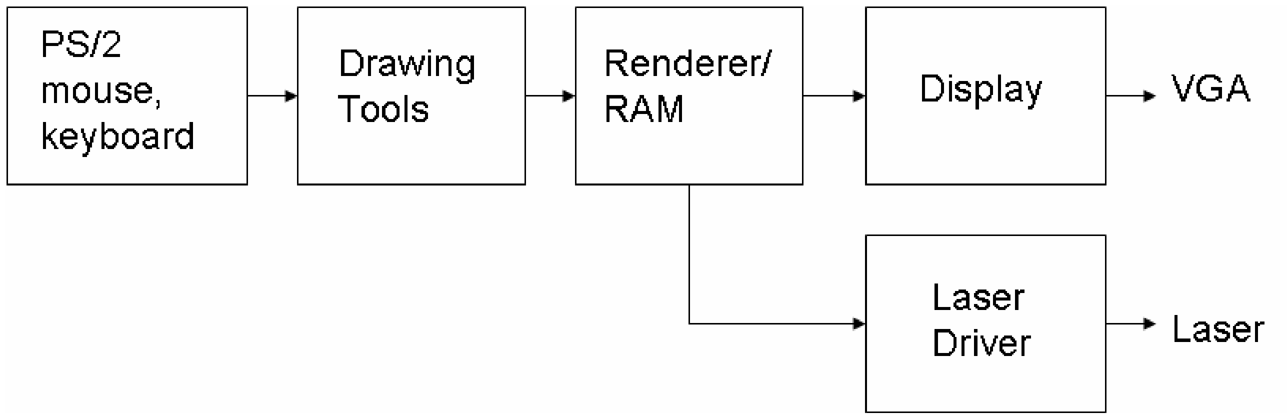

Figure 1. Block diagram of the overall system.

Figure 1 shows the general data flow from the UI to the output devices. The mouse and keyboard send data to the drawing tools, which identify the signals as shape-add or shape-transform commands. These commands are transferred to the renderer/RAM modules, which process the requests and store the shapes. Finally, the VGA and the laser driver display the rendered shapes. All the Verilog modules in this system run on the same reset and clock signals. The reset signal is *reset_sync* that has been passed through a button debouncer, and the clock is a 31.5MHz *pixel_clock*.

2. Description of Subcomponent Modules

2.1. Mouse Interface (m_fsm.v, Figure 2)

The Verilog modules for the PS/2 serial mouse and keyboard interface were written by Profs. Chris Terman and Isaac Chuang in fall 2005. They convert mouse movement and its relative locations into absolute x- and y-coordinates on the VGA screen. The UI's major FSM then takes these coordinates (*mx*, *my*) as inputs for a mapping schematic that designated regions on the VGA screen as one of the four drawing modes: adding a line, adding a circle, translation, or rotation (Figure 3). Depending on which region the clicks land in (indicated by the *clicked_line/circle/trans/rotate* signals), the major FSM calls one of the four minor FSMs that handles each of the drawing modes. These minor FSMs extract the relevant parameters for the shapes and their transformation operations. They each output an enabling signal that tells the transformation modules to start processing the parameters. They also output signals that tell the rendering and RAM modules which shapes and transformations to process.

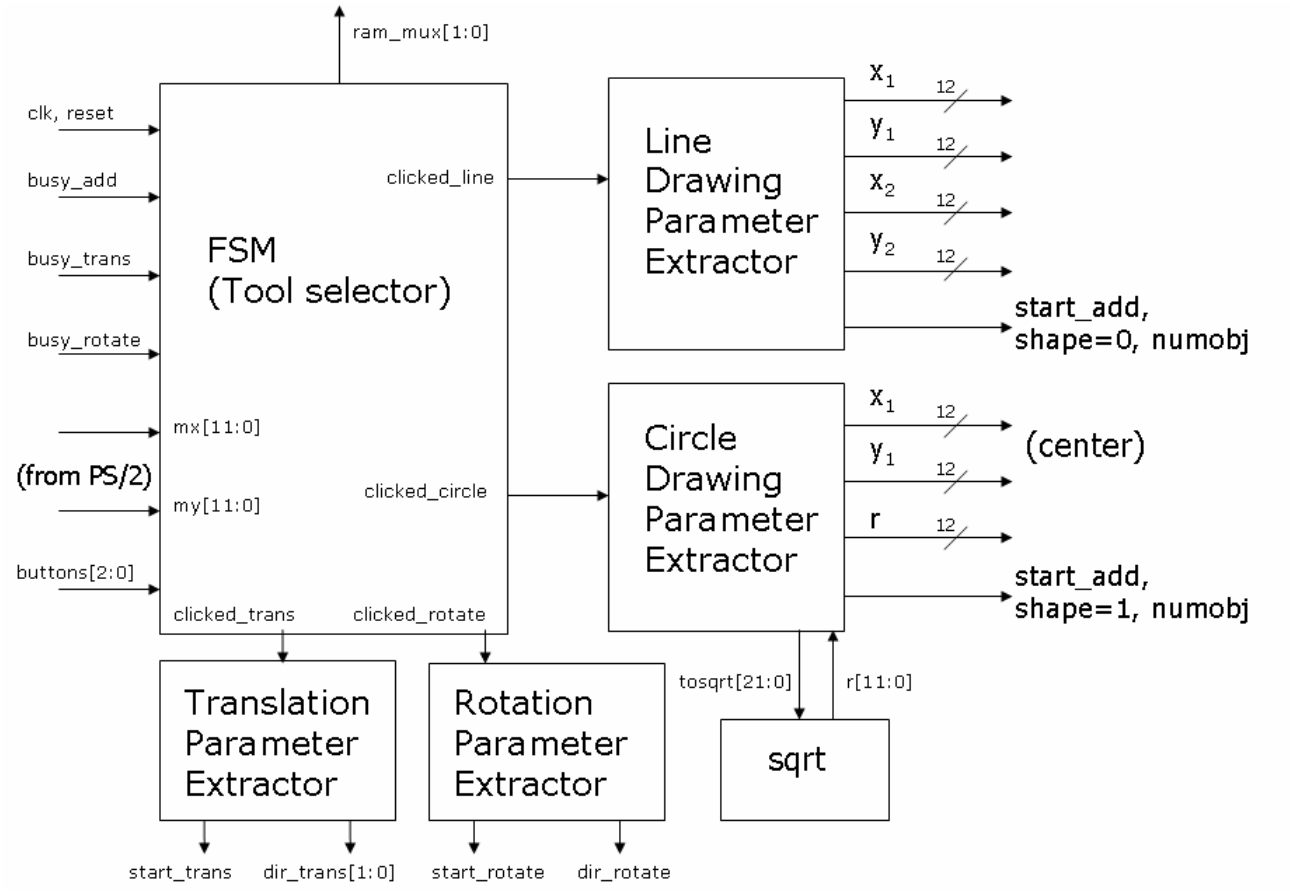

Figure 2. Block diagram for the user interface

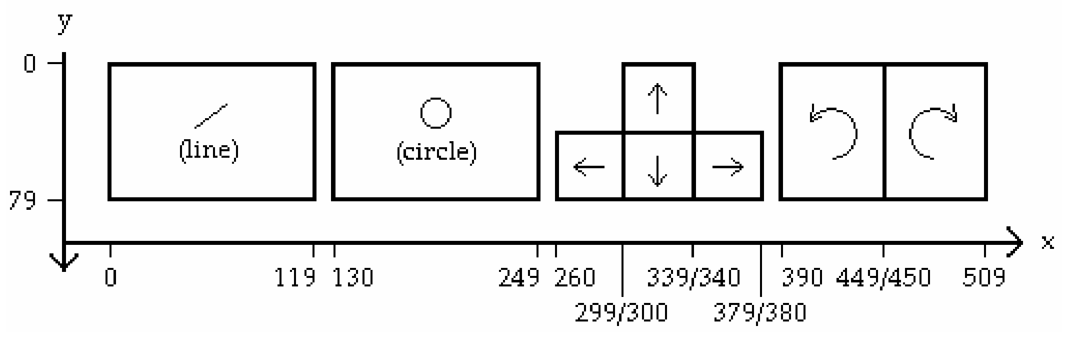

Figure 3. Boundary mapping of pixel values that delineates regions on the VGA screen for various modes.

To draw a line, the user clicks on the line area once and the drawing area twice. The line parameter extractor then outputs the coordinates of the two clicks in the drawing area as the end points of the segments to be drawn. To draw a circle, the user clicks on the circle area and again twice in the drawing area. The circle parameter extractor then outputs the coordinates of the center (first point in the drawing area) and a circumference point (second point). These two shape FSMs also output a 1-bit *shape* signal that equals to 0 if a line is drawn and 1 if a circle is drawn, as well as *numobj* that indicates how many lines and circles there are currently on the screen. The user selects the translation mode by clicking "up", "down", "left", or "right". One can also select rotation by clicking either the clockwise or counter-clockwise arrow. These two modules respectively output a signal that indicates the direction of transformation in addition to the enabling start signals (Figure 2). Each of the minor FSMs contains many wait states in order to make sure that the mouse coordinates are updated, and also to allow time for the rendering modules to communicate with the interface and the RAM.

2.1.1. Minor FSM for Line Parameters Extraction (Figure 4)

If a user clicks in the line region, *clicked_line* will go high and the FSM will go to BEFORE_LINE. BEFORE_LINE counts enough clock cycles (10,000,000) so that the click inside the line area is registered as a line-draw selection. LINE_P1 captures the coordinates of the first endpoint and sets the object identifier signal *shape* to 0. AFTER_LINE_P1 counts clock cycles to register the click of the first endpoint. LINE_P2 captures the coordinates of the second endpoint. AFTER_LINE_P2 keeps looping if the (draw_line.v) module is busy drawing the line, else it increments the object count by 1. FINISH_LINE counts clock cycles to register the coordinates of the second endpoint.

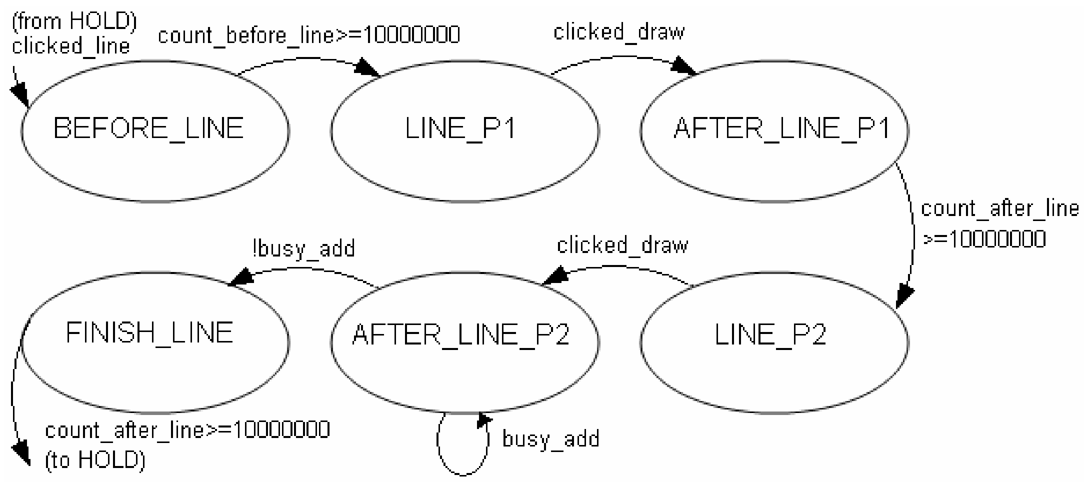

Figure 4. FSM for drawing lines

2.1.2. Minor FSM for Circle Parameters Extraction (Figure 5)

If a user clicks in the circle region, *clicked_circle* will go high and the FSM will go to BEFORE_CIRCLE. BEFORE_CIRCLE counts enough clock cycles (10,000,000) so that the click inside the circle area is registered as a circle-draw selection. CIRCLE_P1 captures the coordinates of the center and sets the object identifier signal *shape* to 1. BEFORE_CIRCLE_P2 counts clock cycles to register the click of the center. CIRCLE_P2 captures the coordinates of a point on the circumference. BEFORE_RADIUS calculates the square of the radius by adding the squares of the distances in the x- and y-directions. This number is fed into a CoreGen module that calculates the square root with a 14-cycle latency. GET_RADIUS keeps looping until the latency period is over. WAIT-CIRCLE keeps looping as the rendering modules are busy drawing the circle, else it increments the object count by 1. FINISH_CIRCLE counts clock cycles to register the coordinates of the point on the circumference.

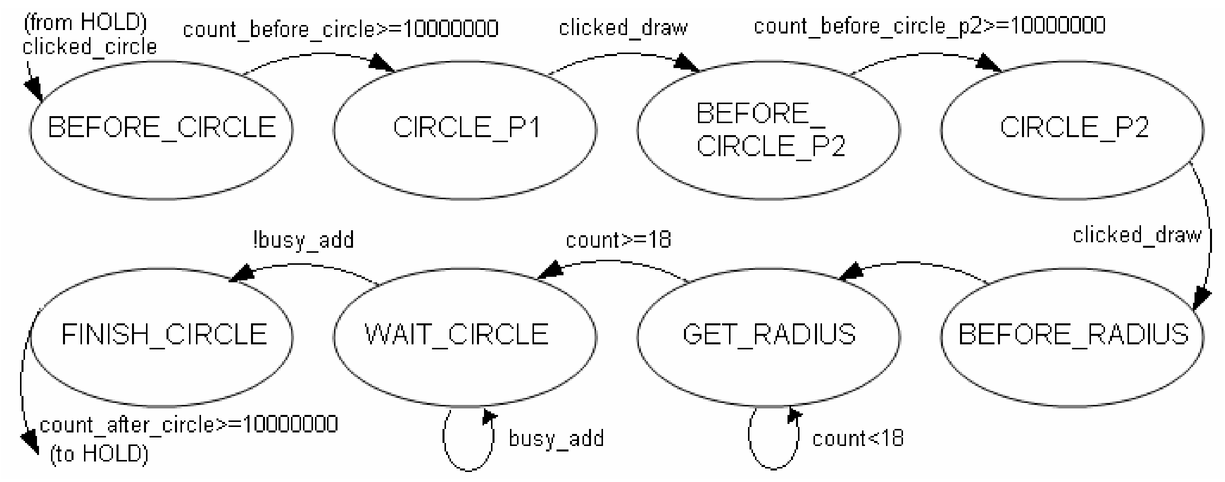

Figure 5. FSM for drawing circles

2.1.3. Minor FSM for Translation Parameters Extraction (Figure 6)

If a user clicks on one of the translational arrows, *clicked_up/down/left/right* will go high depending on which arrow is clicked, and the FSM will go to the corresponding state and set the direction identifier signal (*direction_trans*) accordingly: 00 for UP, 01 for DOWN, 10 for LEFT, and 11 for RIGHT. TRANS keeps looping while the translation module is busy, then FINISH_TRANS counts 10000000 clock cycles to register that the translate operation is completed.

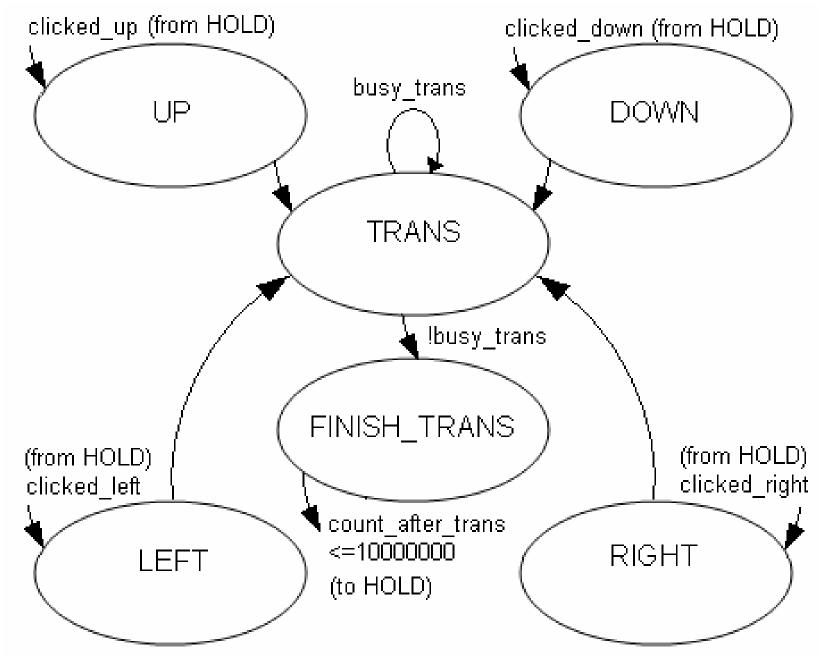

Figure 6. FSM for translation

2.1.4. Minor FSM for Rotation Parameters Extraction (Figure 7)

If a user clicks on one of the rotational arrows, *clicked_rotate_l/r* will go high depending on which arrow is clicked, and the FSM will go to the corresponding state and set the direction identifier signal (*direction_rotate*) accordingly: 00 for ROTATE_L (counterclockwise) and 01 for ROTATE_R (clockwise). ROTATE keeps looping while the rotation module is busy, then FINISH_ROTATE counts 10000000 clock cycles to register that the rotation operation is completed.

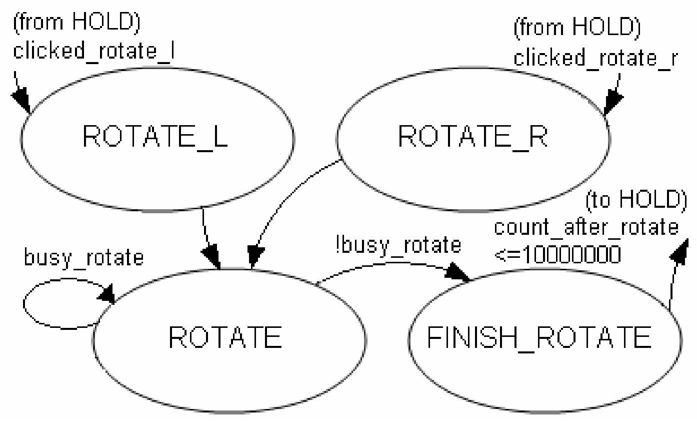

Figure 7. FSM for rotation

2.2. Vector Renderer and RAM

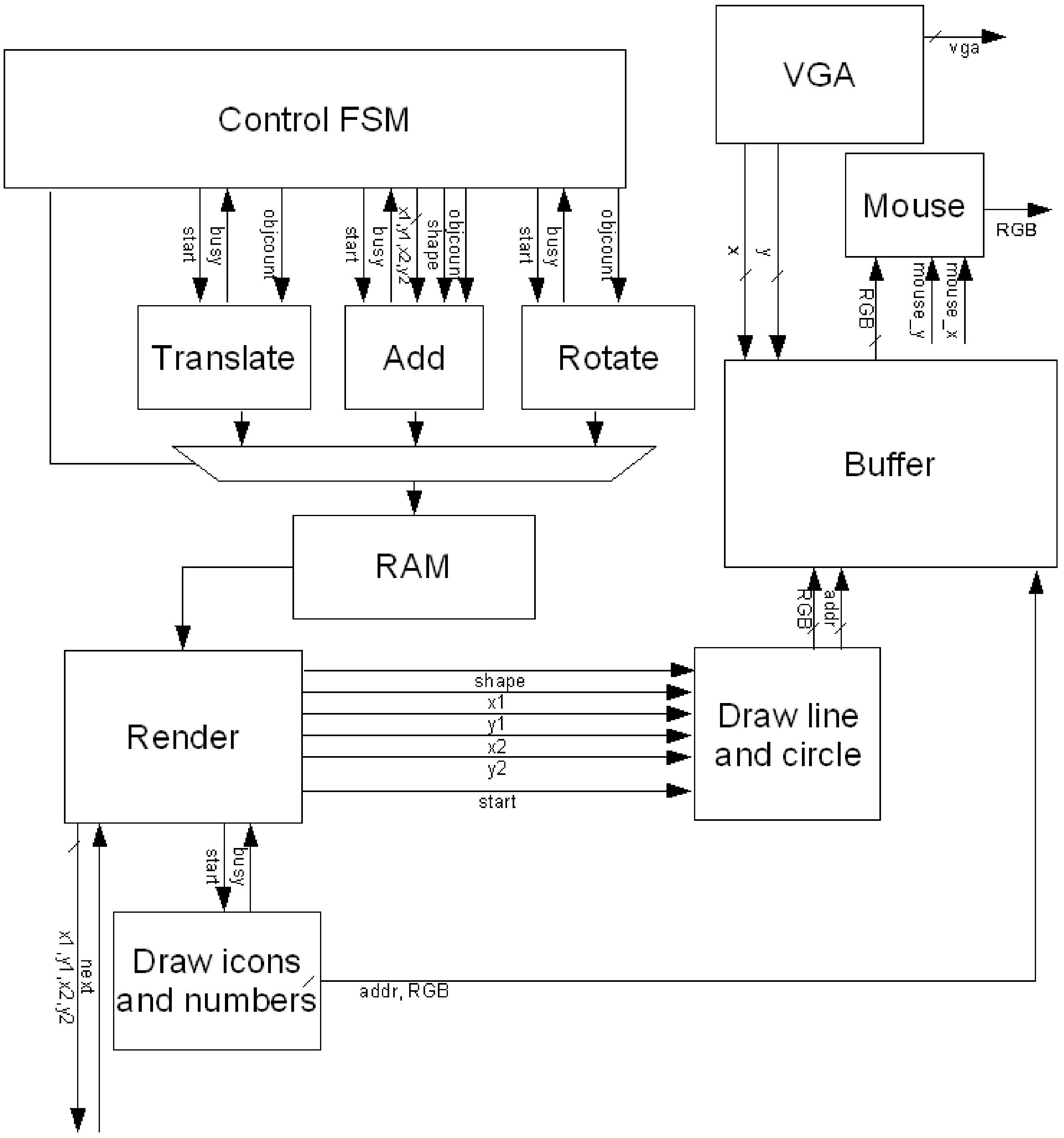

Figure 8. Block diagram of rendering and RAM modules

The purpose of this subsystem is to render a set of lines and circles on the FPGA and represent the data as line segments so the laser driver can sketch the shapes accordingly. These modules also support adding new objects, as well as rotating and translating existing objects.

2.2.1. RAM (ram.v)

The RAM is generated by Xilinx CoreGen and stores the objects being displayed. The data stored is 85-bit wide, with 12 bits for each coordinate, 35 unused bits, 1 color bit, and 1 shape bit that indicates whether the data is for sketching a line or a circle.

2.2.2. Buffer (buffer_ram.v)

We implemented a double buffer so that the data for output to the screen can be read from one buffer, and data for rendering can be written to the other buffer simultaneously. The double buffer is implemented with two block RAMs. The buffer supports three operations: writing to the write buffer, reading from the read buffer, and switching between the two buffers. When the buffers are switched, the buffer module empties the whole write buffer so new data can be written there.

2.2.3. Drawing Lines and Circles (draw_line.v, Figure 9)

This module renders a line or a circle using the Bresenham's algorithm[†], which does not require computationally expensive operations such as division. A circle is approximated with a series of segments as described in the rendering section (2.2.5). This module obtains the coordinates and the shape identifier from the render module and writes the rendering of the objects to the frame buffer. When the coordinates are ready, this module is initiated by the render module. When the draw_line module finishes drawing the line, the busy signal is set to low so that the render module can provide it with data to draw the next object.

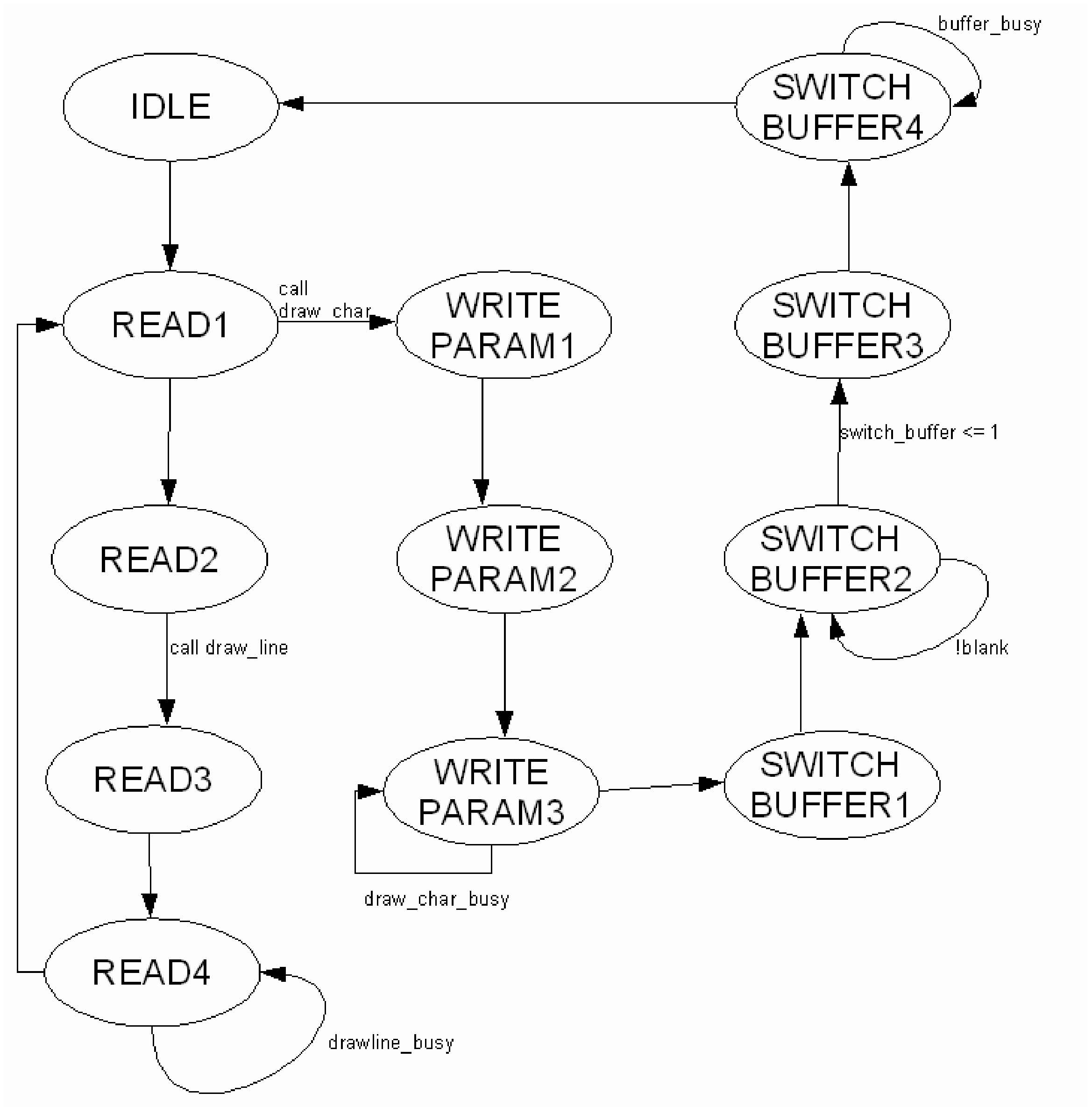

Figure 9. FSM for drawing lines and circles.

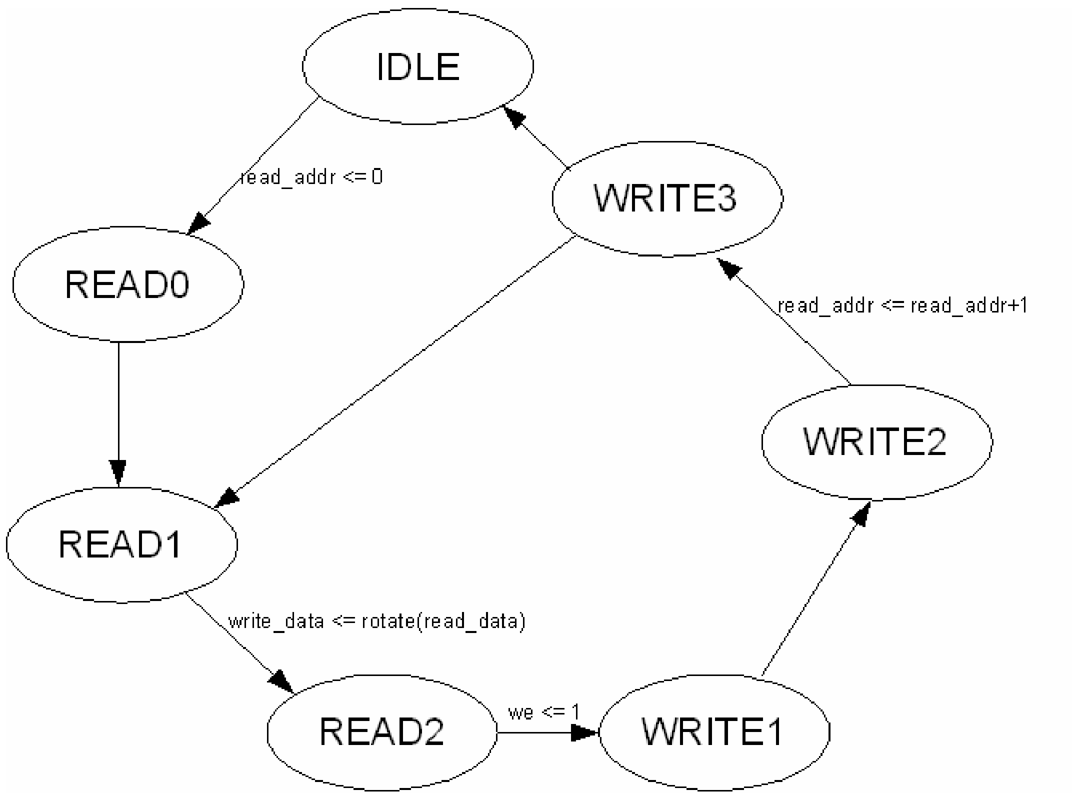

Figure 10. FSM for translation and rotation.

2.2.4. Rotation and Translation (rotate.v, translate.v, Figure 10)

The rotation module applies rotation to the objects. It reads objects from the RAM and writes back the rotated ones. It is implemented as a minor FSM with start and busy signals. The start signal comes from the major FSM in the user interface module. When the rotation module receives the *start* signal pulse, it begins rotating objects by an angle of π/35 and sets the busy signal to high. The rotation is done by multiplying the coordinates with pre-computed constants that correspond to the rotational angle. When the operation completes, the busy signal drops low and the major FSM can then proceed to other operations.

The translation module applies translation to the objects. It reads in the data of the objects from the RAM and writes back the data for the translated objects. It is also a minor FSM with start and busy signals. The start signal comes from the major FSM in the user interface module.

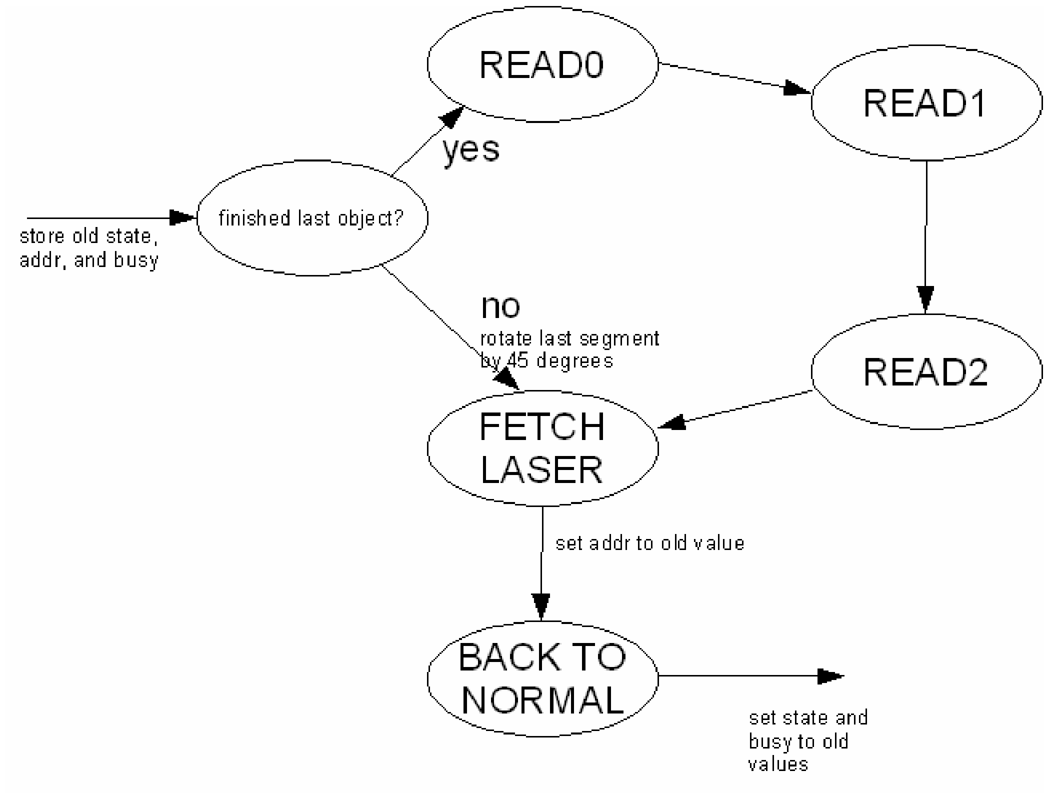
READ0
READ1
yes
finished last object?
store old state, addr, and busy
READ2
no
rotate last segment by 45 degrees
FETCH LASER
set addr to old value
BACK TO NORMAL
set state and busy to old values

Figure 11. FSM for rendering.

2.2.5. Rendering (render.v, Figure 11)

This module reads in data of the objects from the RAM and calls draw_line.v and drawchar.v to render the objects on the screen. It also sends the line segments to the laser driver for projection. Because the laser driver can only draw lines, this module has to approximate circles with octagons and sends the edges in sequential order to the laser driver. During the execution of this module, whenever the laser driver requests the next line segment, it stalls the current work and serves the request. When the module receives a request, it checks whether the current object is sent completely or not. If the object it sent was a line or the last segment of an octagon, then the object is complete and the module reads the next object from the block RAM. If the object is not complete, the module takes the previous segment and rotates it by 45 degrees to obtain the new segment. Rotation is done by multiplying the previous coordinates with pre-computed trigonometric constants that correspond to a 45-degree rotation. When the rendering module finishes processing an object, the add module adds it to the RAM. Displaying the shapes on the VGA screen is handled by the VGA module that generates the sync signals, the line count signal, and the pixel count signal. It is identical to the module in the Pong Game for Lab 4.

2.3. Laser Driver

The laser driver is composed of a digital layer, an analog layer, and a mechanical layer. The digital layer is implemented on the FPGA and makes up the interface between the physical vector drawing engine and digital-to-analog converters (DACs). The analog layer amplifies the signals from the DACs to provide the necessary current (about 500 to 1000 mA) to drive the electromagnetic coils in the galvanometers of the laser scanner (the mechanical layer). The galvanometers are electromagnetic devices that each has a rotor with a permanent magnet suspended by ball bearing inside a cavity (Figure 12). By passing an AC current though the drive coils and a DC current though the biasing coils, the rotor can be made to accurately trace out the AC waveform.

From the point of view of the FPGA, the laser scanner is a black box that moves the laser to a point given by some function of four analog voltages generated by the DACs. These voltages correspond to the movements in the X and Y directions and their bias currents. The laser driver formats the data accordingly for the laser scanner. It then outputs the data to the DACs that set the laser scanner's drive current which powers two rotating orthogonal mirrors. The laser driver will also provide a user interface through the keyboard and VGA in order to adjust image properties such as refresh rate, size, position, rotational angle, and the temperature of the laser diode. This is necessary in order to compensate for the imperfections in the analog drive circuit and mechanical laser scanner.

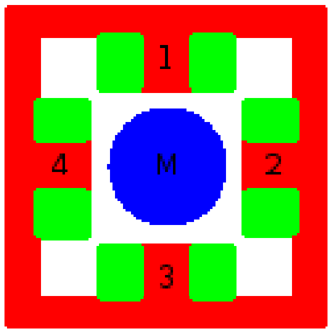

Figure 12. Cross section of a galvo showing the magnet (center) and four coils would around an iron frame.

2.3.1. Digital Layer (Figure 13)

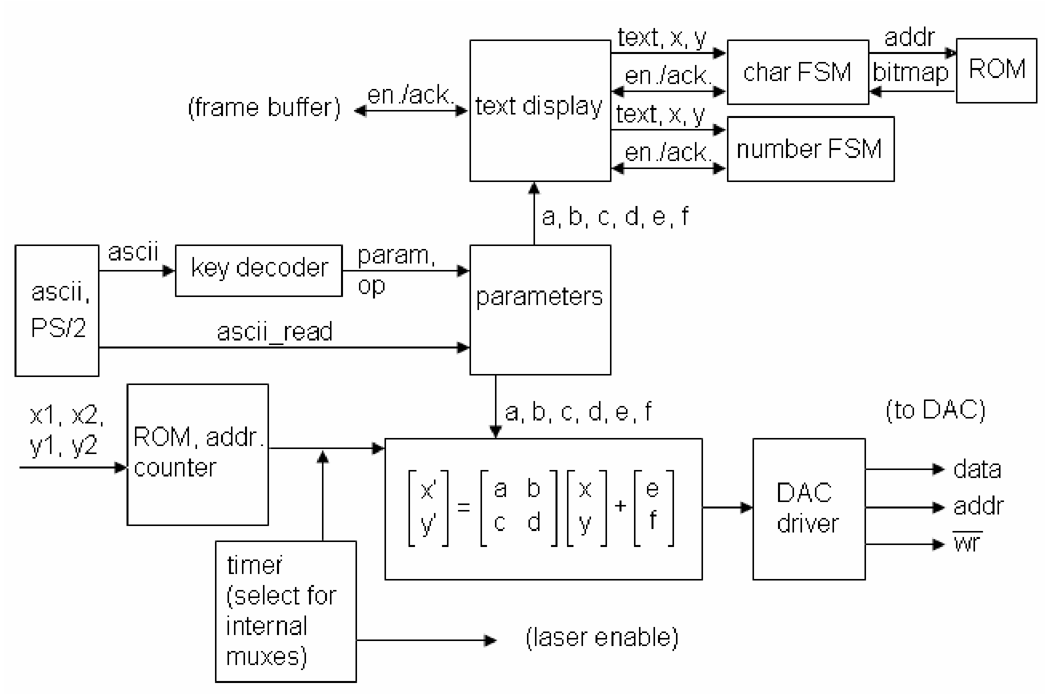

Figure 13. Block diagram of the laser driver.

The main data path in the digital layer is from the vector engine to the DACs. A timer module requests a line segment from the vector engine. The coordinates of input points pass through a linear transformation matrix to scale, shift, or rotate, and the new coordinates are passed to the DACs. The transformation matrix and the period of the timer are stored in a parameters module. In order to modify the laser parameters, we added a PS/2 user interface module (originally written by Prof. Chris Terman, fall 2005). The module takes the input from the keyboard and outputs the current values of all the parameters to the monitor. The interface provides the ASCII value of the key pressed, which the keyboard decoder decodes into an operation to be performed on a specific parameter. The operation and parameter are fed into the parameters module.

The timing module determines the frequency at which points are output to the laser scanner. It allows the refresh rate of the laser display to be chosen by the user or determined automatically by the data it is displaying. Refresh rate is a critical parameter because the laser scanner has limited bandwidth. If the bandwidth is exceeded by a complex image, the output will be distorted. The solution would be to reduce the refresh rate in order to lower the bandwidth.

The parameters from the parameters module are sent out to both the transformation module and a bitmap-based text display engine. To display a number on the VGA, the text display module first converts the number to sign-magnitude (if it is represented as two's complement) and breaks the magnitude into two nibbles. An FSM looks up each nibble in a ROM to get the 35-bit value that correspond to the bitmap for the 5-by-7 pixel character. Another FSM clocks out the bitmap to the VGA frame buffer. The same lookup technique is used to display the icons for the user interface. This entire process is controlled by the frame buffer over start and busy lines.

2.3.2. Analog Layer

The analog layer is composed of two monolithic audio power amplifiers. They output a voltage which is converted to a current by the impedance of the galvanometers' coils. We use current-mode control because it allows for direct manipulation of coil current and therefore the torque on the rotor[5]. Position feedback around the rotor would greatly improve its stability, but we deem the open-loop performance adequate so feedback was not implemented.

2.3.3. Mechanical Layer

The laser scanner is an electromechanical device consisting of two galvanometers (galvos), one for the X direction and one for the Y direction. Each galvo deflects an incoming laser beam with a mirror by a small amount. With proper input, the galvos can be made to trace out shapes with the laser beam.

The galvos have four radial coils spaced ninety degrees apart. At the center, there is a permanent magnet on a shaft (Figure 12). The top and bottom coils are bias coils used to stabilize the permanent magnet in its center position. The left and right coils are drive coils that deflect the magnet from the center position. The magnetic field at the center is the superposition of the bias field and the drive field.

The galvanometers were custom designed and machined, and each one took six to eight hours of machining. Part of the difficulty in machining the galvanometers is that they must be constructed of iron or steel. This presented a unique challenge because the galvanometers needed to be built right the first time.

3. Debugging, Integrating, and Testing

The debugging process involved many ModelSim runs and long waiting time for the code to synthesize. Some problems were readily identified from proofreading the code, such as missing wires and an incorrect implementation of the ram_mux, where the registers were supposed to be wires for combinational logic. One problem found through ModelSim was that the FSM for the mouse UI sometimes shifted states even if there was no mouse click. After the simulated runs exhibiting correct behavior, many problems still cropped up when we synthesized the code. One such problem was that each mouse click added thousands of points or objects that quickly overflowed the RAM buffer, causing nothing to display on the screen. This was because a mouse click lasts about one second and spans millions of pixel clock cycles. The problem was remedied by adding another wait state with a counter that counts to a large number (i.e. 10,000,000) before moving to the next state. This made sure that each mouse click only registered a single point or object.

Implementing the double buffer proved to be a long and arduous task, when the ZBT RAM was still not working after a couple of days of intensive hacking. Block RAM was used instead.

Integrating the code required lots of changing variable names and bookkeeping to make sure all the wires are connected properly before synthesis into the FPGA. Each run took 10 to 15 minutes to complete, but Huy's superb tracking skills helped minimize the wait time.

If the shapes are translated or rotated off the screen, they appear distorted because the transformation modules use unsigned numbers for the parameters such as delta x and y for translation and angle for rotation. When the numbers become negative, they are interpreted as really large numbers which overflow the buffers and result in shape distortion.

4. Conclusion and Acknowledgement

This project has been a tremendous learning experience. We managed to construct a system with both digital and analog electronics of a fair level of complexity. It took a lot of work getting everything together, but we are satisfied with the results. And a big thanks to all the 6.111 staff for helping to make the process less painful and even fun at times.

5. References

[1] Chan. Home Built Laser Projector. http://elm-chan.org/works/vlp/report_e.html.
[2] Norm. Norm's Home Made Laser Show. http://24.202.226.162:81/LaserShow.htm.
[3] Chan, Walker. Analog Laser Light Show. http://bemix.org/analog-laser-light-show.
[4] Griffiths, David. Introduction to Electrodynamics, 3rd ed. Prentice Hall: 1999.
[5] Lundberg, Kent. Feedback Systems for Analog Circuit Design. 2007.

† http://en.wikipedia.org/wiki/Bresenham's_line_algorithm